ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra měření

Akademický rok **2015-16**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	**Filip Schwank**
Studijní program:	**Otevřená informatika**
Obor:	**Počítačové systémy**
Název tématu česky:	**Virtuální senzor pro měření hmotnostního průtoku**
Název tématu anglicky:	**Virtual Sensor for Mass Flow Rate Measurement**

Pokyny pro vypracování:

Navrhněte a s využitím mikrořadiče realizujte virtuální senzor pro měření hmotnostního průtoku vzduchu ve vzduchotechnické jednotce s rekuperačním výměníkem. Pro odhad hodnoty hmotnostního průtoku se budou měřit aktuální teploty uvnitř vzduchotechnické jednotky, jejichž hodnoty se použijí pro výpočet s využitím modelu chování rekuperačního výměníku. Virtuální senzor bude též podporovat připojení řídicích signálů ventilátorů a na základě této informace umožní kontrolovat chování virtuálního senzoru.

Seznam odborné literatury:

[1] Yiu, J.: Definitive Guide to ARMR CortexR-M3 and CortexR-M4 Processors
[2] STMicroelectronics: RM0090 Reference manual
[3] ASHRAE Handbook. SI edition. Atlanta: American Society of Heating, Refrigerating and Air-Conditioning Engineers, 2010, 1 sv. ISBN 978-1-933742-82-3

Vedoucí bakalářské práce:	doc. Ing. Jan Fischer, CSc.
Datum zadání bakalářské práce:	21. ledna 2016
Platnost zadání do[1]:	30. září 2017

[redacted]
Doc. Ing. Jan Holub, Ph.D.
vedoucí katedry

L. S.

[redacted]
Prof. Ing. Pavel Ripka, CSc.
děkan

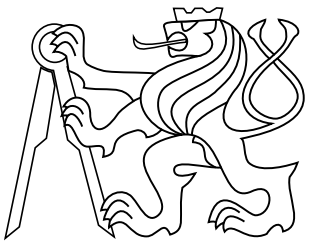
V Praze dne 21. 1. 2016

[1] Platnost zadání je omezena na dobu tří následujících semestrů.

bakalářská práce

Virtuální senzor pro měření hmotnostního průtoku

Filip Schwank



Květen 2016

doc. Ing. Jan Fischer CSc.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra měření

Poděkování

Rád bych poděkoval doc. Fischerovi za jeho rady a vedení v průběhu tvorby této bakalářské práce. Vždy mi ochotně odpověděl na všechny dotazy.

Také bych chtěl poděkovat své rodině a přátelům za jejich podporu a motivaci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Abstrakt

Tato bakalářská práce se zabývá návrhem realizací virtuálního senzoru pro měření hmotnostního průtoku na vzduchotechnických jednotkách. Pro odhad využívá fyzikálního modelu a reálných dat z jednotky a je realizována na mikroprocesorovém kitu firmy ST Microelectronics. Model pracuje se čtyřmi vstupními teplotami, které se měří uvnitř vzduchotechnické jednotky pomocí teplotních senzorů. Dále se měří napěťové úrovně řídicích ventilátorů, které odpovídají určitému hmotnostnímu průtoku a lze je tak porovnat s odhady virtuálního senzoru. Výsledné hmotnostní průtoky jsou zobrazovány na displeji a přenášeny do PC pomocí USB.

Klíčová slova

virtuální senzor; vzduchotechnické jednotky; hmotnostní průtok; výměník tepla; mikrokontrolér

Abstrakt

This bachelor thesis discusses realization of virtual senzor for flow mass measurement in air conditioning units. For estimation uses physical model and real data. It is implemented on a microprocessor kit from ST Microelectronics. The model works with four input temperatures, which are measured inside of the air conditioning unit using temperature sensors. There is also controlling ventilation, that forms certain flow mass, which can be compared with the values from virtual sensor. Calculated flow mass is shown on a LCD display and saved to PC via USB.

Keywords

virtual sensor; air conditioning unit; mass flow; heat exchanger; microcontroller

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Vzduchotechnické jednotky jsou dnes prakticky v každé moderní budově, kde zajišťují optimální klima pro obyvatele. Většinou se jedná o velká zařízení s ohromnými nároky na energie. Proto je důležité, aby se náklady ještě dále nezvyšovaly poruchami, nebo znečištěním prachem a jinými nečistotami. Většinou si vzduchotechnické jednotky s takovými problémy poradí, ale na úkor až mnohonásobného zvýšení spotřeby energie. Aby se zamezilo zbytečnému plýtvání energiemi, jsou nutné časté kontroly zařízení. Ty jsou ale také celkem nákladné, proto je třeba najít nějaké levné řešení.

Pro detekci znečištění je ideální měření hmotnostního průtoku, které zajistí kontrolu jednotky tak, že se bude porovnávat průtok skrze řídicí ventilátory a průtok uvnitř výměníku. Pokud bude průtok na ventilátorech větší než ten uvnitř, je jednotka znečištěná a vyrovnává tento problém zvýšením výkonu ventilátorů.

Senzory pro měření hmotnostního průtoku mohou být drahé, avšak je tu možnost využít nepřímého měření pomocí virtuálního senzoru, který bude hmotnostní průtok odhadovat z fyzikálního modelu klimatizační jednotky, případně i z experimentálních dat.

Pro řízení virtuálního senzoru bude použit mikrokontrolér, který zajistí veškeré výpočty, komunikace se senzory, případně komunikaci s počítačem, kam bude odesílat naměřená a vypočtená data.

2 Rozbor problematiky

Většina moderních budov dnes potřebuje vzduchotechnické jednotky pro udržení komfortního prostředí uvnitř. Tato zařízení mají ohromný výkon, tudíž i velké spotřeby energií. V kancelářských budovách je podíl celkové spotřeby energie ve vzduchotechnických jednotkách téměř 50%, v hotelích a obchodních centrech se spotřeba energie pohybuje okolo 40 % a v nemocnicích okolo 30% [1]. Tyto náklady se však mohou mnohonásobně zvýšit vlivem prostředí, kdy se například zadře klapka rekuperačního výměníku. Závada je ale z hlediska využití skrytá, protože nemá vliv na komfort uvnitř budovy. Vzduchotechnická jednotka je tedy nucena zvýšit příkon ventilátorů, případně musí zvýšit výkon topicí, nebo chladicí soustavy. Nárůst není tak markantní, ale z dlouhodobého hlediska se jedná o neefektivní provoz.

Protože je žádoucí mít náklady co nejnižší, je dobré zavést nějaký systém detekce poruch. Na to existují různé metody FDD (Fault Detection and Diagnostics). FDD je oblast zabývající se automatizací procesů pro odhalování nestandardního chování systémů a určování příčin takového chování. Nevýhodou těchto metod je potřeba velkého množství dat a tedy i velkého množství senzorů. Vzhledem k tomu, že ani moderní vzduchotechnické jednotky nejsou dostatečně vybaveny senzory určenými pro diagnostiku (většinou obsahují minimální množinu senzorů potřebných k řízení systému), nabízí se zde řešení v podobě využití virtuálních senzorů [1].

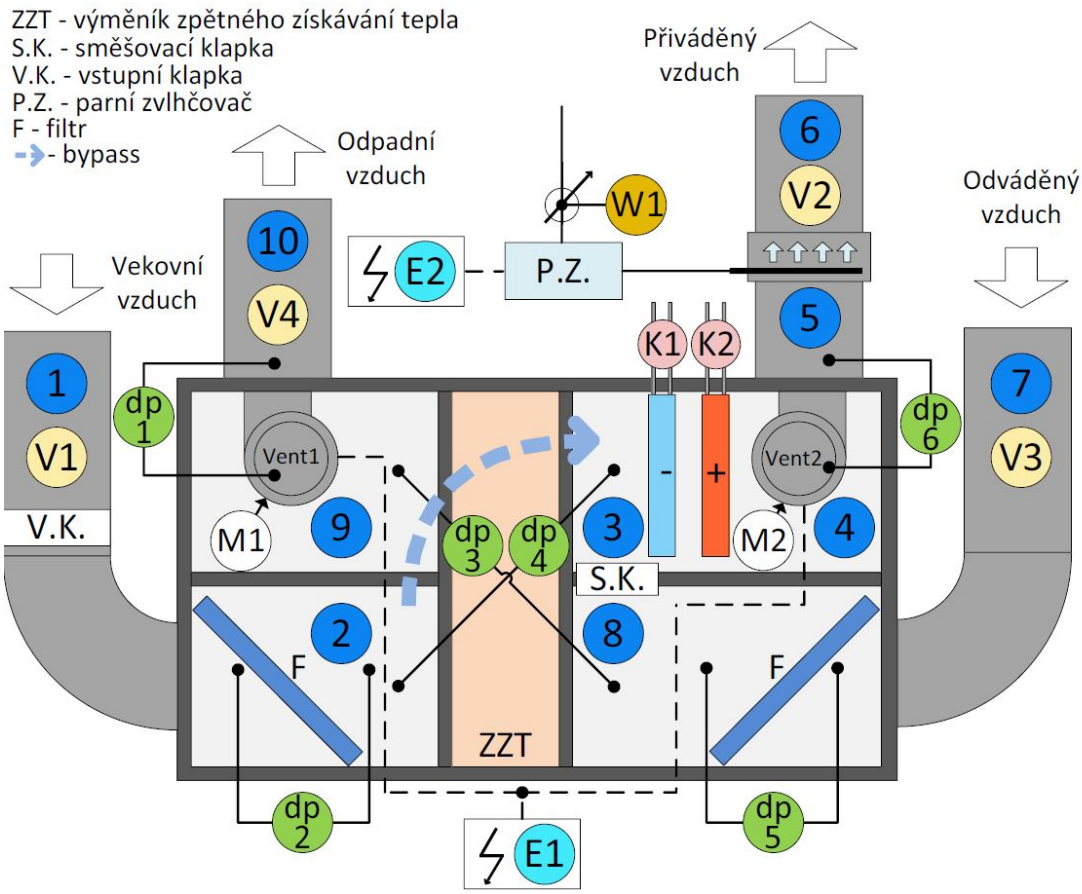
Virtuální senzor VS (Virtuální senzor) odhaduje danou veličinu na základě údajů, které jsou dostupné z jiných zdrojů informací. Jako zdroj informací může posloužit například fyzikální model zařízení nebo je možné využít jiné dostupné senzory. VS často pracují s informacemi z řídící elektroniky, s historií stavů jednotky a s měřenými veličinami. Pro nasazení VS je nutné znát co možná nejpřesnější popis technického vybavení a uspořádání vzduchotechnické jednotky [1].

2.1 Výpočet hmotnostního průtoku pomocí VS

Detekovat znečištění uvnitř vzduchotechnické jednotky lze pomocí virtuálního senzoru, který měří teplotu na 4 místech uvnitř výměníku. Následně lze z teplot vypočítat hmotnostní průtok vzduchu dle charakteristického modelu vzduchotechnické jednotky, který se porovná s nastavením ventilátorů.

Průtok vzduchu skrze ventilátory je řízen napěťovými úrovněmi a pokud vypočítaný průtok neodpovídá průtoku vzduchu skrze ventilátory, lze vzduchotechnickou jednotku považovat za neefektivní. Díky tomuto systému detekce poruchy je možné včas objevit neefektivní provoz a snížit tak náklady.

Pro výpočet hmotnostního průtoku využiji VS na principu tzv. „gray-boxu“. Existuje více dělení virtuálních senzorů, ale to je nad rámec této práce. Tento princip spojuje „white-box“ (modelem řízený VS – využívá fyzikální zákony systému) a „black-box“ (daty řízený VS – modelování založené na empirických vztazích bez znalosti fyzikálních dějů systému). Použitý VS využívá k odhadu hmotnostního průtoku vzduchu předem odměřenou křivku účinnosti tepelného přenosu rekuperátoru (TTE – Temperature Transfer Efficiency). [1].



Obr. 1 Schéma experimentální jednotky [1]. Výpočet hmotnostního průtoku pomocí VS – teplotní senzory označené modře čísly 2, 3, 8, 9.

Následující rovnice popisují výpočet hmotnostního průtoku na experimentální jednotce na fakultě stavební ČVUT v Praze. Využívá se 4 teplotních senzorů umístěných uvnitř výměníku. Senzory označené 2 a 3 jsou v přívodu vzduchu před a po ohřevu odpadním vzduchem. Senzory 8 a 9 potom jsou v odvodu vzduchu, který je ochlazován přívodním (obr. 1) (tím se také šetří energie spojené s provozem).

$$\vartheta_S = \frac{t_3 - t_2}{t_8 - t_2} \tag{1}$$

$$\vartheta_E = \frac{t_8 - t_9}{t_8 - t_2} \tag{2}$$

$$TTE = \frac{m_S}{m_{min}} \cdot \vartheta_S = \frac{m_E}{m_{min}} \cdot \vartheta_E \tag{3}$$

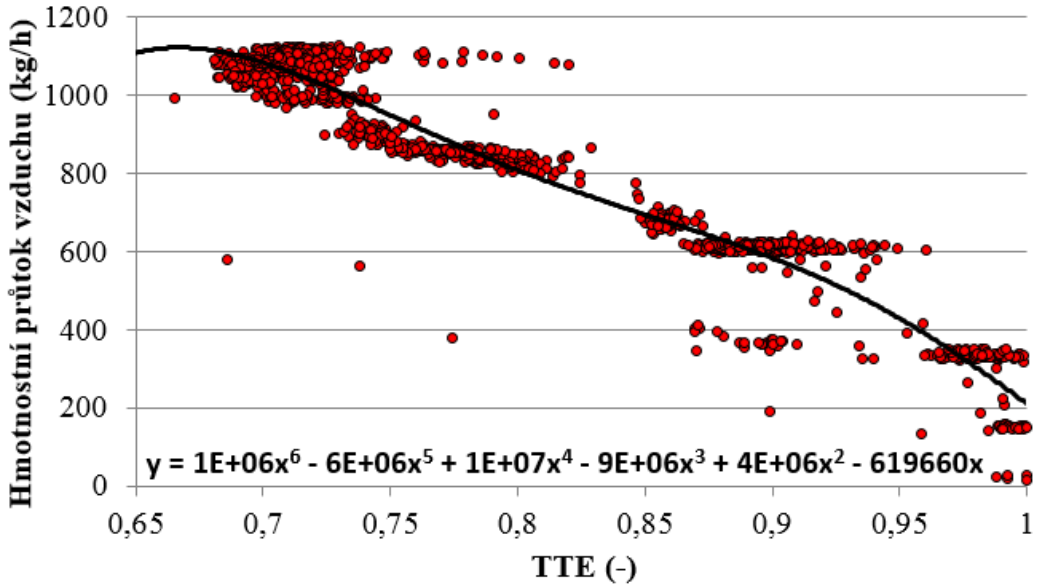
$$m_E = \frac{1}{f}(TTE)\Big|_{m_S \leq m_E} \quad m_S = m_E \frac{\vartheta_E}{\vartheta_S}\Big|_{m_S > m_E} \tag{4}$$

$$m_E = m_S \frac{\vartheta_S}{\vartheta_E}\Big|_{m_S \leq m_E} \quad m_s = \frac{1}{f}(TTE)\Big|_{m_s > m_e} \tag{5}$$

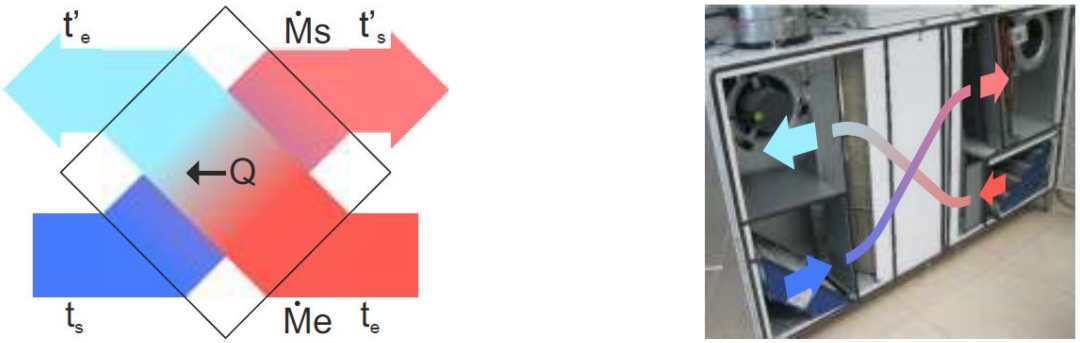
Teploty označené t_2, t_3, t_8, t_9 jsou získané z teplotních senzorů na výměníku. ϑ je potom poměr rozdílu teplot. S označuje přiváděný vzduch a E odváděný (obr. 3). Dále TTE jsou naměřené hodnoty účinnosti tepelného přenosu rekuperátoru (obr. 2) (hodnoty TTE nejsou měřeny přímo, ale pomocí odvozeného vztahu [1, rovnice 8]). f je model výměníku - charakteristický polynom (tab. 1), který reprezentuje fyzikální vlastnosti jednotky. Nakonec tedy lze získat z naměřených teplot hmotnostní průtoky přiváděného a odpadního vzduchu (m_S a m_E).

p_1	=	-88.97
p_2	=	334.5
p_3	=	13.27
p_4	=	-1152
p_5	=	570.6
p_6	=	1269
p_7	=	-676.6
p_8	=	-444.1
p_9	=	-56.45
p_{10}	=	836.6

Tab. 1 Koeficienty polynomu pro výpočet hmotnostního průtoku na vzduchotechnické jednotce.



Obr. 2 Naměřené hodnoty *TTE*, které se použijí pro výpočty měření VS.



Obr. 3 Schéma a umístění rekuperátoru [1],t jsou jednotlivé teploty,M – hmotnostní průtok, S označuje přiváděný vzduch a E odváděný.

3 Realizace virtuálního senzoru

Pro realizaci měření hmotnostního průtoku pomocí VS bude třeba nějaké levné a poměrně jednoduché řešení, aby bylo možné kontrolovat stav, případně zaznamenávat hodnoty z výměníku. Realizace tohoto virtuálního senzoru je především určena do vzduchotechnických jednotek, které nemají vůbec žádné, nebo jen minimální senzorové sítě.

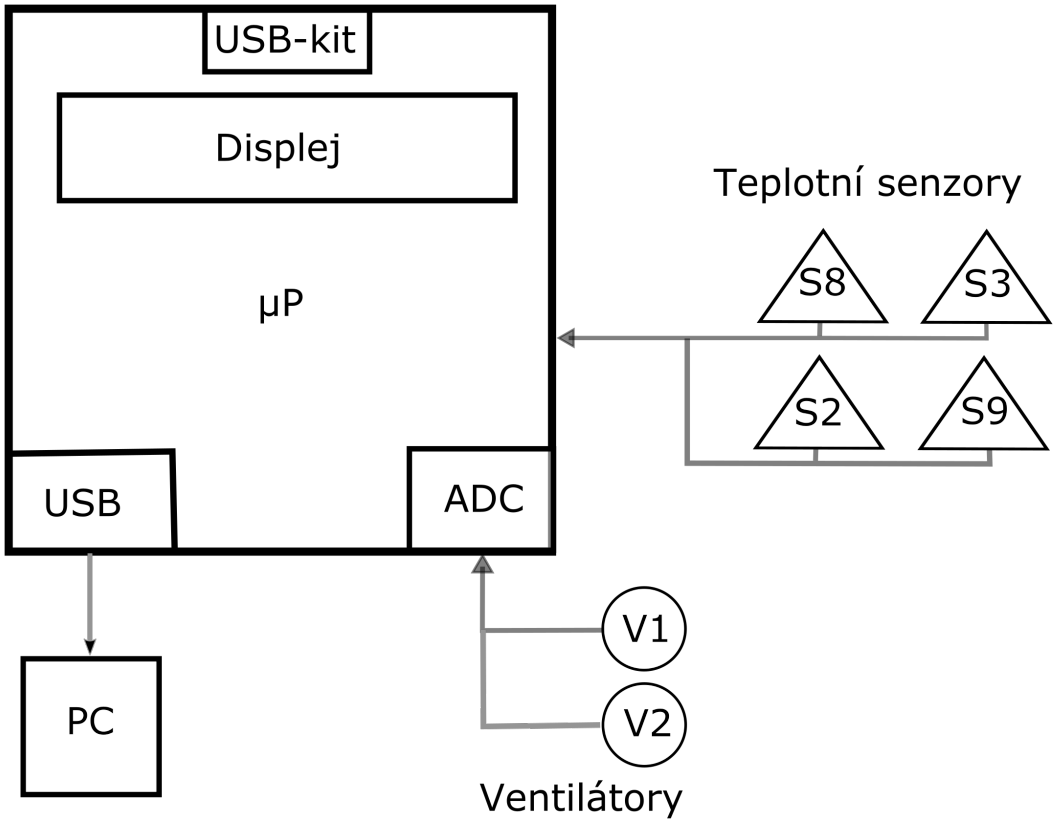
Hmotnostní průtok je vypočítáván ze 4 teplot (kap. 2.1), které jsou měřeny na různých místech uvnitř výměníku (obr. 1). Pro toto měření použiji digitální senzory firmy Maxim Integrated (dříve Dallas), DS18B20. Tyto senzory komunikují na Onewire protokolu, mají vlastní AD převodník, takže posílají již digitalizovaná data.

Pro digitální teplotní senzory budu také potřebovat mikrokontrolér, který s nimi bude komunikovat a naměřená data dále zpracuje, případně odešle do PC. V dnešní době jsou dobrým řešením mikroprocesorové kity, které standardně mají vstupní brány, AD a DA převodníky, případně další periferie. Jelikož mám zkušenosti s kity od firmy ST Microelectronics, tak jsem vybral takový, který obsahuje zobrazovací 6 znakovou jednotku. Dále má vestavěné rozhraní USB a periferie, které jsem zmiňoval. Jedná se o STM32L-Discovery kit s procesorem STM32L152RB (ARM Cortex-M3 – 128 KB Flash, 32 MHz CPU). Pro potřeby tohoto projektu by měl mít dostatek funkcí.

Projekt si z programovacího hlediska mohu rozdělit na několik dílčích částí, které budu postupně realizovat:

1. Založení projektu a výběr knihoven,
2. zobrazovací a ovládací periferie – zobrazování dat na vestavěný LCD displej a ovládání např. pomocí tlačítka,
3. komunikace se senzory pomocí Onewire protokolu,
4. výpočet hmotnostního průtoku z naměřených teplot uvnitř vzduchotechnické jednotky,
5. napěťové úrovně ventilátorů v rozmezí 0-10V – pomocí ADC získám digitální hodnotu napěťové úrovně na ventilátorech,
6. odesílání naměřených a vypočtených dat pomocí USB do PC,
7. PC aplikace pro záznam a zobrazení dat z VS (ukládání do textového souboru).

Prvotní verzi programu a zapojení budu realizovat na nepájivém poli, abych si ověřil všechny funkce. Obrázek níže nastiňuje ideové propojení kitu s měřenými jednotkami a výstup do počítače (obr. 4).



Obr. 4 Blokové schéma projektu. Mikroprocesor použit pro komunikaci se senzory (S2, S3, S8, S9) a měření napětí na ventilátorech (V1, V2). Dále je znázorněn výstup do PC pomocí USB.

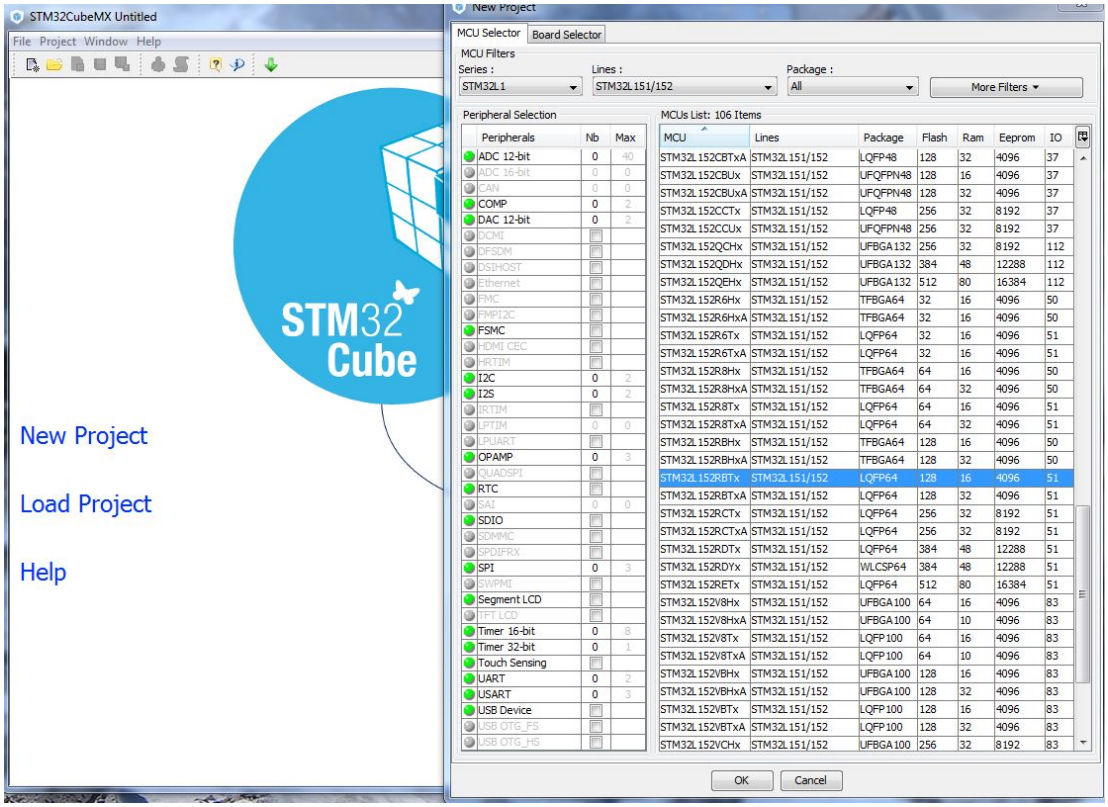
3.1 Knihovny - STM Cube MX

Programovat aplikace pro kity od ST lze několika způsoby. V ARM asembleru, kde je třeba znát detailně registry procesoru, nebo v jazyce C. V C je programování snazší, avšak stále je nutné rozumět registrům, pokud nejsou použity knihovny. Programovat těmito způsoby jsem se naučil v předmětu *Návrh vestavěných systémů*. Nyní u tak rozsáhlého projektu není tolik kladen důraz na programování, ale na realizaci aplikace. Proto mohu využít knihoven.

Knihoven existuje jistě celá řada, mezi nejpoužívanější patří Peripherals a Cube přímo od ST Microelectronics. Peripherals jsou jen takový „obal“ registrů, zatímco Cube poskytuje nejen základní ovládání periferií, ale také komplexnější funkce pro jednoduché využití USB, nebo displeje.

Pro práci na projektu jsem použil knihovny STM Cube MX firmy ST Microelectronics. Balík obsahuje také grafickou aplikaci, která v mnohém usnadňuje práci, neboť jsou schematicky zobrazuje všechny periferie a umí k nim dle nastavení vytvořit inicializační soubory.

Postup pro začátek je jednoduchý. Nejprve se založí projekt, pro který se zvolí kit, na kterém aplikace poběží (obr. 5). Vybrat lze kit – vybrání kitu rovnou nastaví některé periferie (také Cube ví co se na kitu nachází), nebo přímo procesor – což je jistější volba, protože si to programátor nastaví jak potřebuje.



Obr. 5 Výběr jednotlivých procesorů pro založení projektu v Cube (označen procesor použitý v projektu).

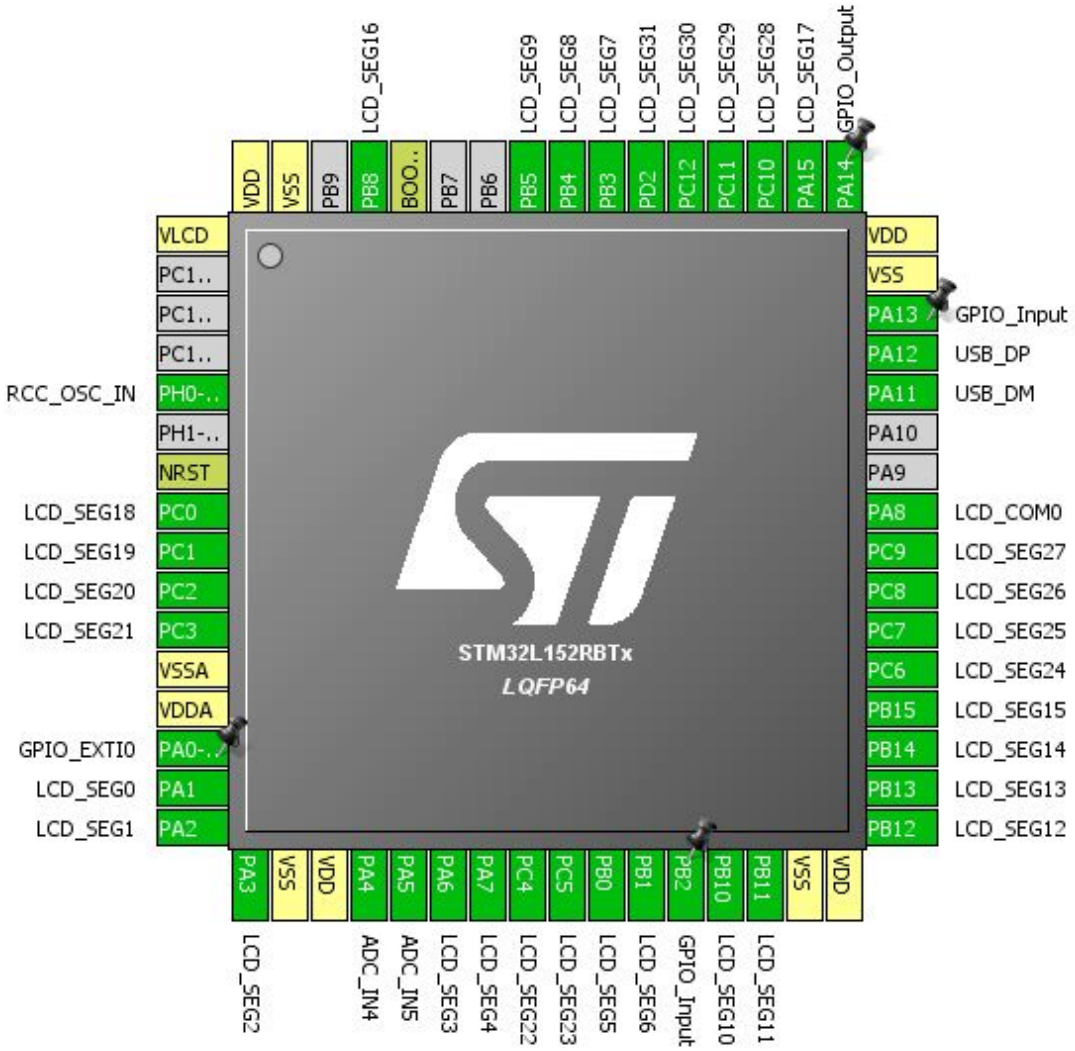
Dále se objeví úvodní obrazovka, kde jsou zakresleny piny kolem procesoru vybraného kitu a jejich nastavení. V levém panelu lze nastavit jednotlivé periferie, v mém případě jsem aktivoval celý LCD displej, USB v CDC módu, dva ADC kanály a nastavil vstupní a výstupní piny. Potom stačí zadat vygenerování projektu a objeví se celá aplikace včetně inicializačních kódů pro jednotlivé periferie.

Aplikace má mnoho dalších funkcí, vždy závislých na zvoleném kitu. Například nastavení hodin pro jednotlivé periferie a procesor, dále detailní zobrazení jednotlivých komponent kitu, kde je možné snadno nastavit každý registr.

Po vygenerování projektu do požadovaného vývojového prostředí (jsem zvyklý na Keil μVision 5, ale Cube podporuje také další vývojová prostředí) lze programovat aplikaci ve vyznačených uživatelských oddílech a využívat balík funkcí z HAL knihoven, které jsou součástí Cube.

Hardware Abstraction Layer (HAL) poskytuje programátorovi jednodušší manipulaci s periferiemi, kdy není tolik třeba znát přesné registry. Avšak pokud dojde nějakému chybnému nastavení ze strany uživatele, je stále nutné registrům rozumět, neboť ladění programu pracuje pouze s registry. HAL má další výhodu v tom, že pokud by bylo nutné kód portovat na nějaký jiný kit, tak je do jisté míry zajištěna kompatibilita.

3.2 Využití pinů na procesoru STM32L152RB



Obr. 6 Nastavení vybraných pinů kitu v aplikaci Cube.

Před zapojením součástek je třeba se nejdříve podívat, na který pin procesoru mohu kterou součástku umístit a také, jak jsou obsazené periferie. Také, pokud se budou napájet některé součástky 5V, musí být piny, na které se tyto součástky připojí tzv. 5V-tolerantní. To se dá zjistit z dokumentace procesoru, kde jsou jednotlivé piny podrobně vypsány [2].

Na teplotní senzory je třeba 1-2 piny (kap. 3.4.1), na ventilátory 2 piny. Dále bude využito USB (USB zabere 2 piny na kitu – data+ a data-) a LCD panel. Obr. 6 popisuje rozložení jednotlivých použitých pinů na kitu. LCD panel jich zabírá nejvíce, protože každý segment má 16 částí, z toho 4 obsadí vždy jeden pin. Segmentů je 6, takže 24 pinů, ale ještě je třeba započítat společné COM připojení, která jsou 4 –tedy celkem 28 pinů pro LCD.

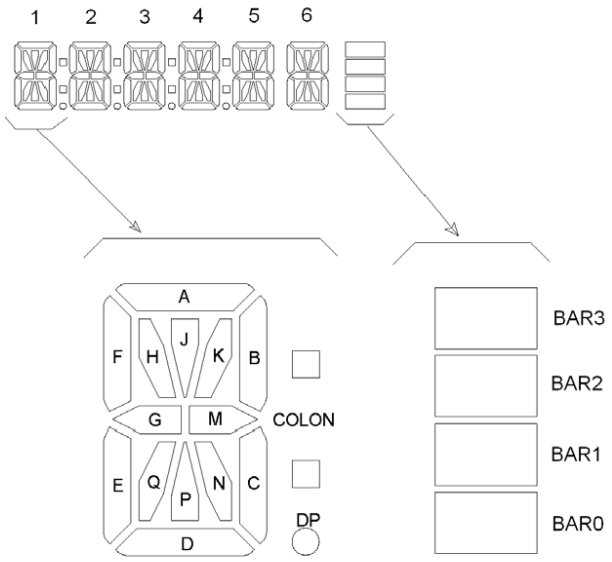
Dle dokumentace jsem tedy pro senzory vybral pin PB2, který byl v první verzi obousměrný, pro druhou verzi 2 piny – PA13 (vstup), 14 (výstup) (viz. zapojení senzorů 3.4.1). Všechny jsou 5V-tolerantní. Pro ventilátory zbyly pouze 2 piny s ADC kanály (viz. kapitola o ventilátorech 3.6). USB má piny jasně dané pod označením PA11,12. Kvůli USB ale musím přivést do procesoru externí oscilátor, který je umístěný u ladícího procesoru (pin PH0 – RCC_OSC_IN) [2].

3.3 Zobrazovací a ovládací funkce

Pro začátek je třeba zprovoznit zobrazovací funkce, aby bylo možné hodnoty přímo kontrolovat. Jelikož se budou zobrazovat teploty, napěťové úrovně ventilátorů, případně nějaké vypočtené hodnoty, tak by mi mělo 6 segmentů na vestavěném displeji stačit (teploty jsou maximálně do 125° C, napětí do 10V, průtoky řádově až tisíce). Ovládat se potom bude pomocí tlačítka, které zobrazované hodnoty posune na další.

3.3.1 Vestavěný LCD displej kitu

Vybraný kit obsahuje vestavěný LCD displej (obr. 7), který se dělí na 6 segmentů a „bar“ zobrazovač (4 vodorovné obdélníky jako poslední segment). Celkem tedy zapnutí displeje zabere na kitu 24 pinů pro jednotlivé segmenty a 4 piny pro společné COM (společné signály – pixel svítí, pokud příslušný COM a segment mají maximální amplitudu) [3].



Obr. 7 Vestavěný LCD zobrazovač na kitu pro zobrazení až šesti 14-segmentových znaků, čtyř „barů“, desetinné tečky (DP) a dvojtečky (COLON) [4].

Díky knihovnám lze zprovoznit displej velmi snadno. Soubor který jsem použil (`stm32l152c_discovery_glass_lcd.h`) je přiložen v balíku Cube. Stačí zavolat funkci `BSP_LCD_GLASS_Init()` , která zobrazovač inicializuje a pak už například pro zobrazení nějakého textu stačí použít funkci `BSP_LCD_GLASS_DisplayString(uint8_t* ptr)` .

Drobný problém byl, když jsem chtěl využít zobrazení desetinné tečky u čísel (segment DP). Musel jsem tedy modifikovat výše uvedenou zobrazovací funkci, aby znak tečky umístila na DP a ušetřila tak vlastně jeden zobrazovací segment. Také pokud je desetinná tečka na více jak 4. segmentu, tak se zobrazí pouze jako mezera (velká čísla nepředpokládám a navíc budou odesílána ve vyšší přesnosti do PC), protože displej nemá na těchto místech DP a funkce nezobrazuje znak tečky na segmentu (místo DP tam jsou BAR registry).

3.3.2 Ovládání přístroje

Na displeji se budou zobrazovat hodnoty, které se naměří. Protože je zobrazovač malý, hodnoty se budou střídat automaticky, případně zásahem uživatele. Proto využiji uživatelské tlačítko na kitu, nebo přidám externí tlačítko.

Díky knihovnám je práce s tlačítkem jednoduchá, zvolil jsem nastavení tlačítka, které vygeneruje přerušení do procesoru. Pin PA0 (na kterém je uživ. tlačítko) je napojen na EXTI0 (External interrupt/event controller), který detekuje náběžnou hranu a při zmáčknutí tlačítka vygeneruje přerušení. Celkem je těchto linek 16, každá číselně odpovídá pořadí pinu pro různé brány. Pomocí NVIC (Nested vectored interrupt controller) se zachytí přerušení a zavolá se jeho obsluha.

V Cube se tlačítko nastaví na EXTI0, v NVIC povolí jeho přerušení. Pro obsluhu přerušení je potom třeba použít funkci `HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`, kde se provede potřebná operace. Tedy v tomto případě se po stisku tlačítka přepne zobrazení dat na další položku.

3.4 Teplotní senzory DS18B20

Pro měření tedy použiji 4 digitální senzory DS18B20 (2.1). Komunikace probíhá na Onewire protokolu, který je založen na Master-Slave modelu. Master – tedy v mém případě STM kit řídí veškerou komunikaci na datové lince. Využívá se čtecích a zapisovacích slotů a resetovacích pulzů.

Senzory mohou být zapojené v parazitním módu, kdy se připojí jen zem a datový pin. Datový pin se potom použije jako zdroj. Toto zapojení je výhodné, pokud je nutné šetřit kabely. Nevýhodou ale je, že Master musí po dobu konverze držet linku v 1 (napěťové úrovni), aby byl senzor dostatečně napájen.

Druhá možnost, kterou použiji, protože zde není nutné šetřit na každém kabelu, je využití všech tří pinů - napájení, zem, data. V tomto módu nemusí kit parazitně napájet senzory. Napájení může být v rozmezí 3-5.5V[5].

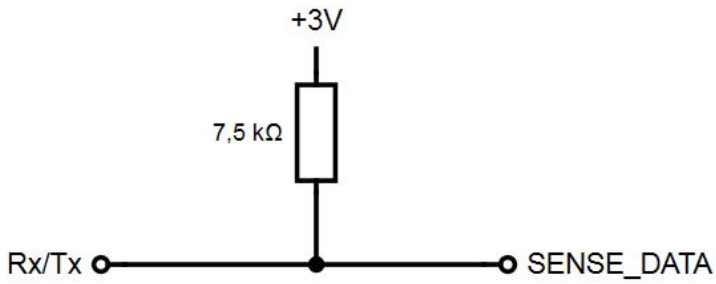
V obou módech je však třeba použít nějaký pull-up rezistor, neboť linka je v klidovém stavu v 1. Dle datasheetu je doporučeno alespoň 4.7kΩ. Pro jistotu jsem použil 7.5kΩ rezistor.

Z hlediska programování této části aplikace byl největší problém s časováním. Nejdříve jsem chtěl využít TIM časovače na kitu, bohužel se mi ale nedařilo je úplně správně nastavit. Resp. jsem potřeboval, aby běžely vždy krátkou chvíli – řádově mikrosekundy. Opětovné spouštění a zastavování trvalo moc dlouho, v jiné verzi se zase nechtěl vymazat příznak přerušení, nebo některé registry.

Nakonec jsem se rozhodl, že procesor v tak krátkém časovém úseku stejně nemusí nic vykonávat – čekal jen na přerušení, nebo v jiném pokusu „polloval“ příznak přetečení čítače. Proto jsem vytvořil jednoduchou softwarovou čekací smyčku – for-cyklus, jehož vykonání zabere dostatek času v mikrosekundách. Při taktu procesoru 24MHz by tedy za jednu mikrosekundu měl procesor vykonat 24 instrukcí. Senzory jsou naštěstí relativně tolerantní, takže nebyl problém (s drobnou dopomocí náhledu osciloskopem) správně nastavit čekací smyčku.

3.4.1 Zapojení teplotních senzorů

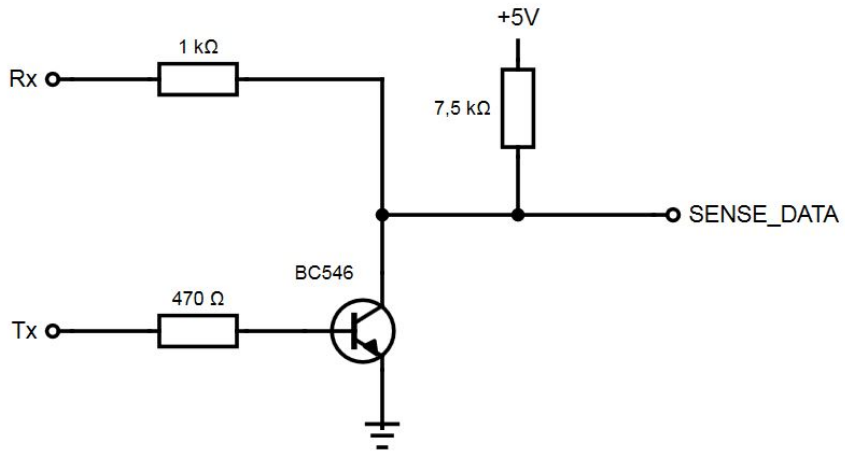
V první realizaci komunikace se senzory byl použit jeden datový pin na kitu, který zajišťoval jak příjem, tak vysílání dat na linku a přepínal tedy vstup na výstup a naopak (obr. 8).



Obr. 8 První způsob zapojení senzorů bez ochrany kitu – napájení jen 3 volty, pull-up rezistor pro jistotu o něco větší – 7.5kΩ.

Druhý způsob už ale počítá s tím, že senzory budou vedeny kabely na různá místa vzduchotechnické jednotky, proto je nutné zajistit ochranu kitu před nežádoucími vlivy, jako je například impulz statické elektřiny při dotyku člověka s kontaktem, konektorem apod.

Je tedy nutné rozdělit komunikaci na pin pro příjem a pin pro vysílání. Vstup stačí ochránit zařazením rezistoru v řádech několika kΩ do série, mezi pin a datovou linku teplotních senzorů. Výstup bude budit NPN tranzistor připojený bází k pinu kitu, kolektorem na datovou linku a emitorem k zemi. Při sepnutí tedy procesor uzemní linku (obr. 9).



Obr. 9 Druhý způsob zapojení s rozděleným vysílacím a vstupním pinem.Ochranné rezistory (vstup 1kΩ, výstup – kolektor NPN tranzistoru BC546 470Ω). Napájení již na plném 5V napětí.

3.4.2 Onewire komunikace se senzory

Master vždy začíná komunikaci na lince, kde nemusí být jen teplotní, ale i další typy senzorů komunikující pomocí Onewire protokolu.

Komunikace začíná vždy resetovacím pulzem – to znamená, že Master stáhne linku k zemi, senzor pak detekuje logickou 0. Tento pulz musí trvat alespoň 480 mikrosekund, na který senzor odpoví do 15 až 60 mikrosekund prezenčním pulzem trvajícím 60 až 240 mikrosekund.

Dalším stavebním kamenem komunikace se senzory jsou čtecí a zapisovací sloty. Každý zapisovací slot musí trvat alespoň 60 mikrosekund. Pro zápis logické 1 se linka stáhne do nuly na 15 mikrosekund, pro logickou 0 je linka stažena po celý slot. Slave, tedy senzor čte po aktivaci slotu, tj. po 15 mikrosekundách.

Čtecí slot probíhá obdobně. Master aktivuje slot stáhnutím linky do nuly alespoň na jednu mikrosekundu, Slave potom odpoví logickou 1 tím, že nechá linku v napěťové úrovni, nebo logickou 0 – stáhne linku k zemi. Master pak jen přečte data po minimálně 15 mikrosekundách od začátku slotu.

3.4.3 Příkazy a vyhledání senzorů

Když už byla zprovozněna základní komunikace, je třeba zavést dle Onewire protokolu několik příkazů. Ty se dělí na funkční a ROM příkazy.

ROM příkazy zajišťují identifikaci jednotlivých senzorů pomocí jedinečné 64 bitové ROM adresy.

Funkční příkazy se používají ke čtení obsahu paměti uvnitř senzorů, zahájení teplotní konverze atd.

ROM příkazy

Na začátku se musí nejdříve určit, s kým Master bude komunikovat. To zajišťují ROM příkazy. V případě, že je na lince přítomen jen jeden senzor, lze použít příkaz Skip ROM, který přeskočí vyjednávání o adresaci jednotlivých senzorů. Protože se nemusí určovat jeden senzor ze skupiny, tak je možné rovnou použít funkční příkazy.

Pokud je na lince více senzorů, je nutné nejprve určit s kým se bude komunikovat. Ještě existuje jedna výjimka, kdy se použije příkaz Skip ROM a pak příkaz pro konverzi, který přikáže všem senzorům zahájit teplotní odměr. Kdyby po Skip ROM následoval jiný příkaz, došlo by ke kolizi – každý ze senzorů by chtěl začít komunikovat s Masterem, protože si myslí, že byl osloven.

Pro určení jednoho senzoru, se kterým bude v danou chvíli probíhat komunikace, je třeba, aby se Master nejprve naučil všechny jedinečně 64bitové ROM všech senzorů na lince. Pro to existuje příkaz Search ROM, který je z celého Onewire protokolu nejkomplexnější, proto ho ještě více popíšu dále. Jakmile se Master naučí všechny ROM zadá příkaz Match ROM a odešle na linku jednu z adres, tím se určí jediný senzor pro komunikaci, ostatní čekají na reset pulz. Po vyhledání a adresaci je možné použít funkční příkazy.

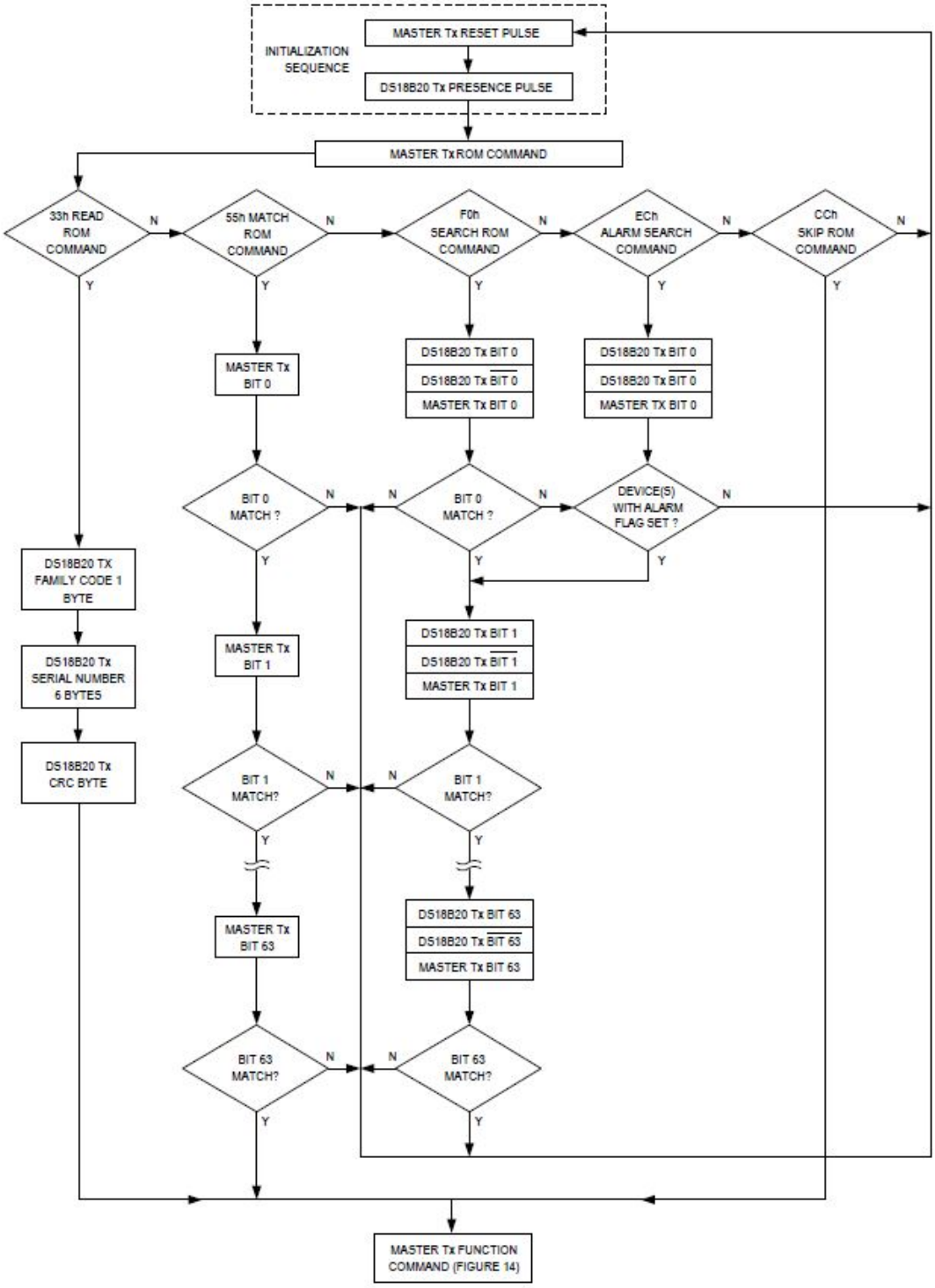
Search ROM Při programování tohoto algoritmu na vyhledání jednotlivých senzorů jsem postupoval dle diagramu v datasheetu (obr. 11), na který odkazovala dokumentace senzorů. Bohužel se mi nepodařilo tento algoritmus řádně zprovoznit, našel jsem tedy jiný zdroj, kde už byl algoritmus hotový, jen jsem ho upravil, aby fungoval v mém kódu [6].

V podstatě zde jde o to, aby Master eliminační metodou jednoznačně určil každý senzor na lince. Zároveň si tedy zapisuje jednotlivé bity ROM a také body posledního rozdílu.

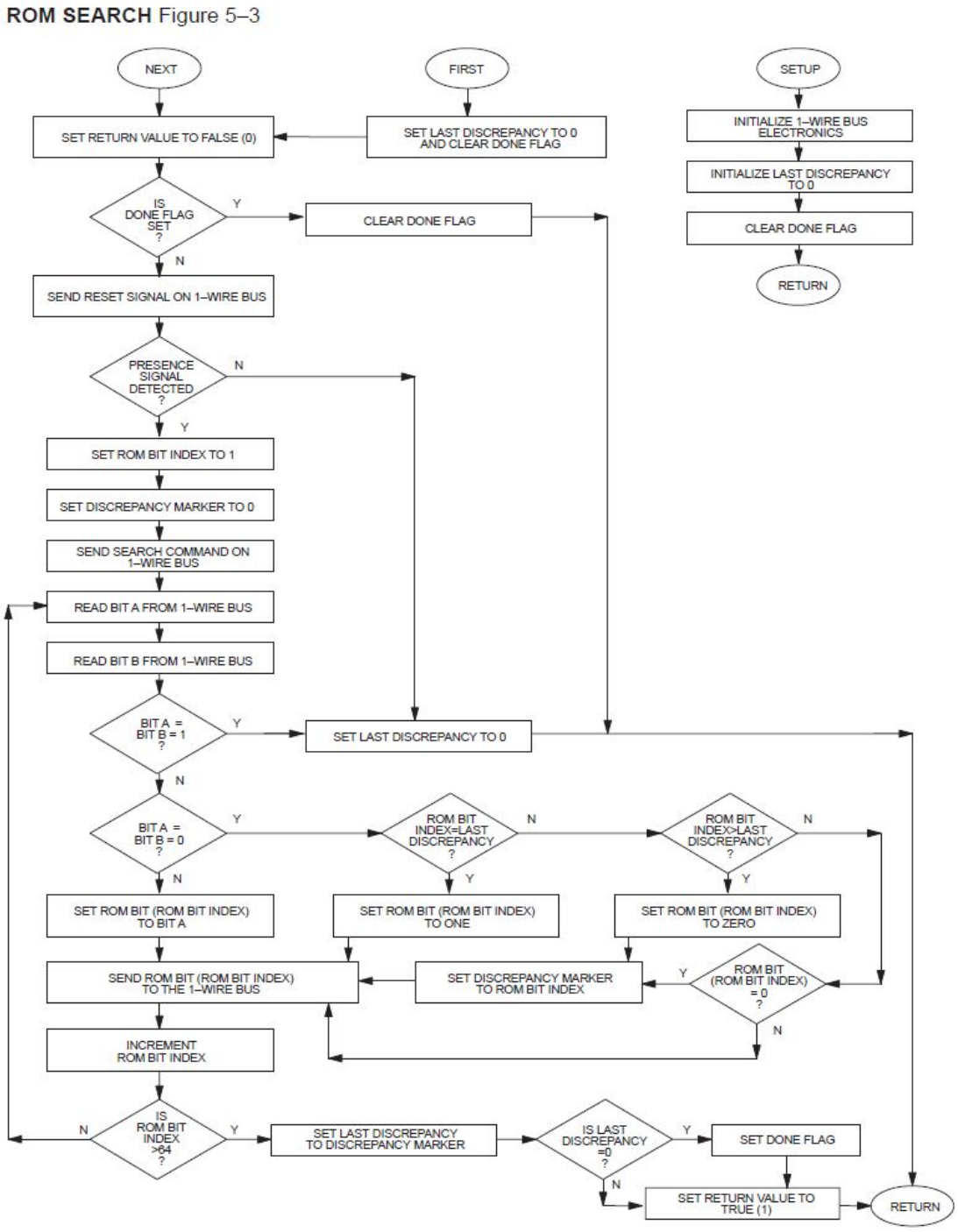
Učení začíná tím, že Master vyšle příkaz Search ROM, kdy každý senzor vyšle první bit své ROM, následně vyšle také jeho negaci. V případě, že jsou přítomny jen senzory s prvním bitem pouze 0, nebo pouze 1, Master přečte 01 resp. 10 a není kolize.

Pokud je na lince kombinace bitů, tak je přečteno 00 (bitový AND), vznikne kolize a Master si musí zapamatovat toto místo, aby v dalším průchodu odeslal potvrzovací 1, v prvním se odesílá 0. Tím, že Master zapíše určitý potvrzovací bit, se vyřadí vždy druhá skupina senzorů, která pak čeká na další průchod, tedy na reset pulz.

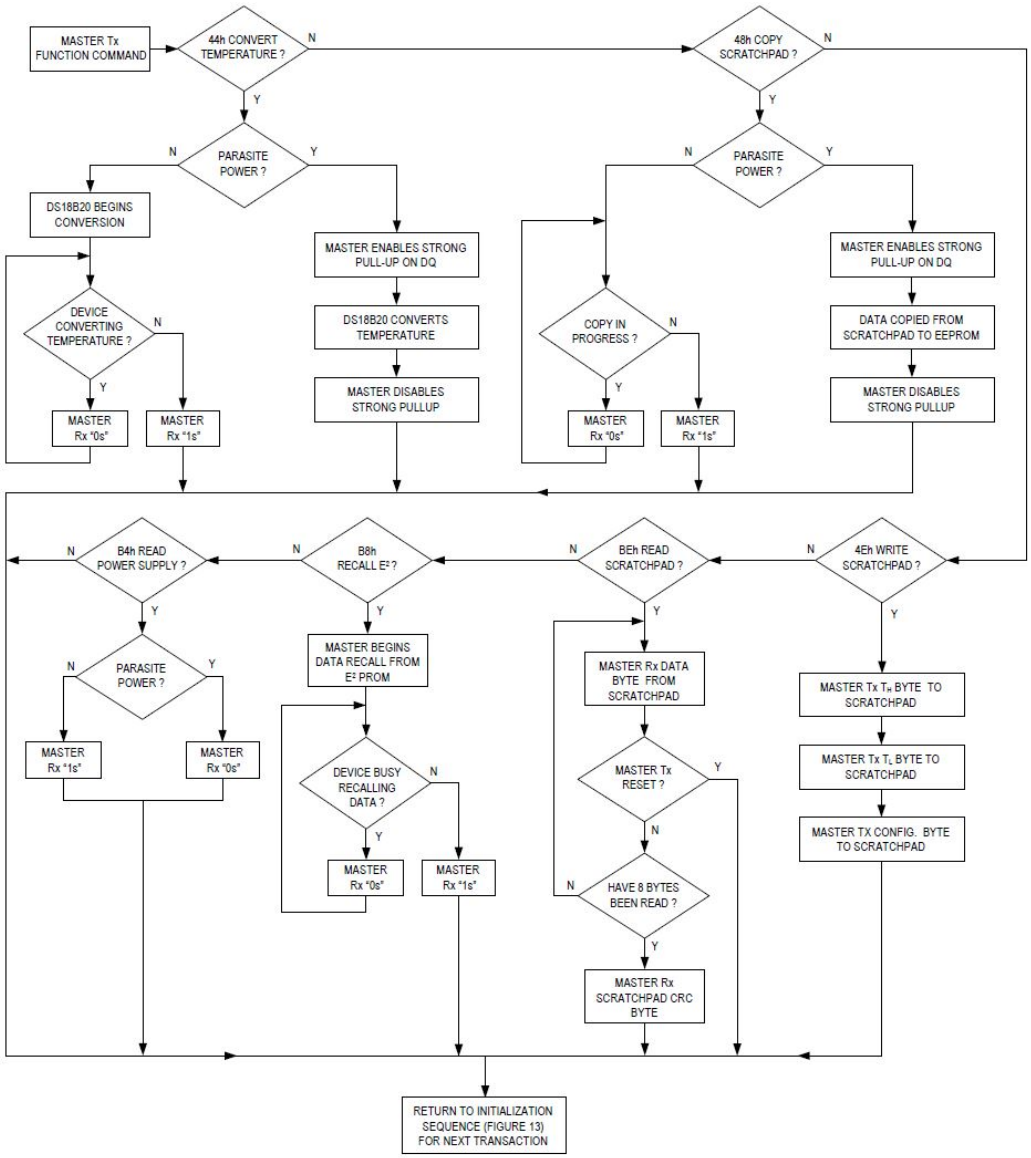
Tímto způsobem Master postupně přečte všechny bity, vyšle reset, zadá opět vyhledávací příkaz, určí další senzory a celý proces se opakuje, dokud jsou detekovány nějaké senzory na lince.



Obr. 10 Rozhodovací strom ROM příkazů Onewire protokolu [5].



Obr. 11 Algoritmus pro vyhledání všech senzorů na datové lince [7].



Obr. 12 Rozhodovací strom funkčních příkazů Onewire protokolu [5].

Funkční příkazy

Každému funkčnímu příkazu musí předcházet ROM příkaz. Pouze v případě, že je na lince jediný senzor, tak po Skip ROM lze využít všech funkčních příkazů. S více senzory lze po přeskočení použít pouze měření teploty.

Mezi funkční příkazy patří ConvertT, který vyšle určitému, případně i všem senzorům (dle zvolené ROM) příkaz zahájení teplotní konverze. Ta trvá dle zvoleného nastavení (výchozí je nejdelší 12 bitová konverze) 90 - 750ms.

Dále je třeba přečíst tato naměřená data pomocí ReadScratchpad, pro tento příkaz je už nutné číst z jednoho určitého senzoru, jinak vznikne kolize (více senzorů by odesílalo svá data). Ve výchozím 12bit módu tedy Master obdrží 9 bajtů dat, tj. teplotní data, také nastavení senzoru a CRC součet.

Pro účely tohoto projektu stačí přečíst první dva bajty dat, kde jsou umístěna teplotní data. V prvním bajtu jsou 4 celočíselné a 4 desetinné bity a ve druhém 5 znaménkových a 3 celočíselné bity. Tedy 8 bitů na celé číslo, 4 desetinné bity a bit určující znaménko.

Existují i další příkazy, ale ty nebudou potřebné pro tuto aplikaci. Případně by se mohl využít příkaz pro nastavení na 8bit konverzi, ale vzhledem k tomu, že systém výměníku je pomalý, tak si mohu dovolit delší konverzi a tím získám přesnější údaje pro výpočty.

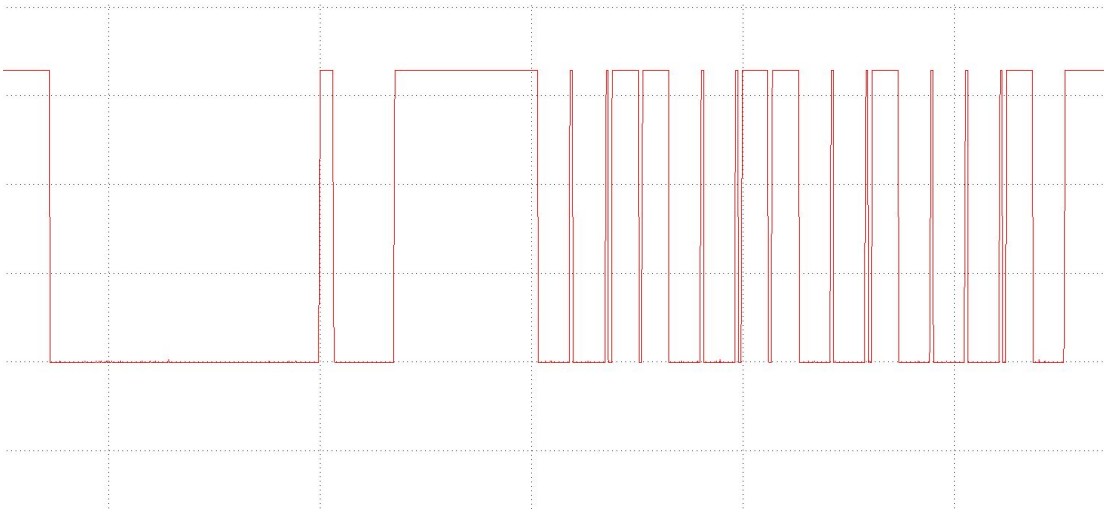
3.4.4 Příklad komunikace

Následuje příklad komunikace, který je aplikovaný v tomto projektu. Tabulka je podobná jako v dokumentaci senzorů [5].

Master MÓD	DATA	Popis
Tx	Reset	Reset pulz.
Rx	Prezence	Senzor odpovídá na reset prezenčním pulzem.
Tx	Search ROM	Následuje celý algoritmus vyhledání, opakovaný, dokud nejsou nalezeny všechny senzory
...	...	...
Tx	Reset	Reset pulz.
Rx	Prezence	Senzor odpovídá na reset prezenčním pulzem.
TX	Skip ROM	Přeskoč identifikaci
Tx	ConvertT	Zahaj konverzi teplot
...	...	Počkej aspoň 750ms
...	...	Následující transakce 4x
Tx	Reset	Reset pulz.
Rx	Prezence	Senzor odpovídá na reset prezenčním pulzem.
Tx	MatchROM	Zavolej výběr ROM
Tx	64bitROM	Odešli zvolenou 64bitROM
Tx	ReadScratch	Požádej o naměřená data
Rx	až 9 bajtů dat	Master čte celou pamět senzoru.

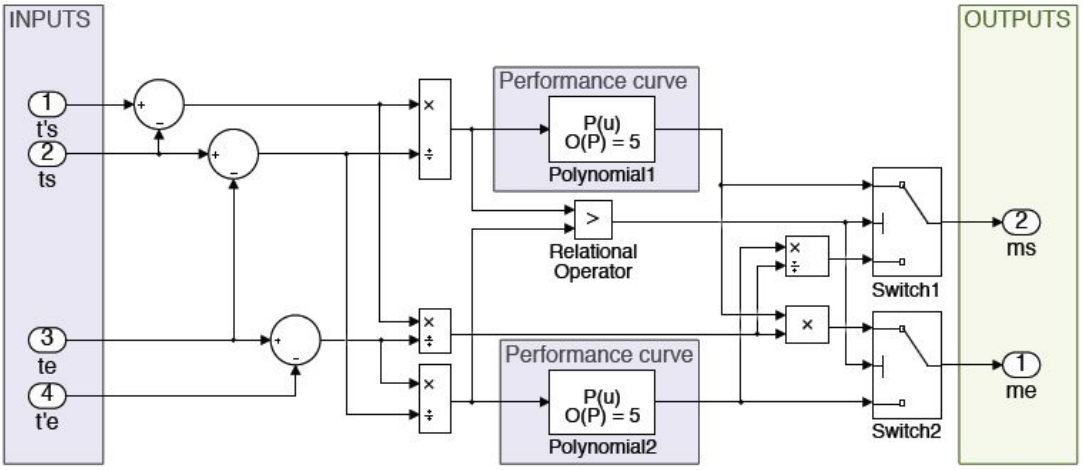
Tab. 2 Příklad komunikace z mé implementace Onewire protokolu. Na lince jsou 4 senzory ve standardním zapojení (všechny 3 piny). Nejprve probíhá učení ROM adres, dále pak teplotní konverze a jednotlivé adresace pro přečtení naměřených hodnot.

Také jsem pomocí Instrulabu na kitu Nucleo-F411 zachytil část komunikace se senzory. Protože je komunikace velmi rychlá, je třeba Instrulab nastavit na 8bit rozlišení a 2 Mega Samply za sekundu (2MSPS). Na obrázku (13) je zachycena část komunikace, kdy Master vyslal reset pulz a jakmile linku pustil, tak mu senzor po chvíli odpověděl prezenčním pulzem. Potom začalo vyslání příkazu – je vidět 16 vyslaných bitů.



Obr. 13 Zachycená komunikace se senzory. Vyslán reset a příkaz.

3.5 Výpočet hmotnostního průtoku na kitu



Obr. 14 Blokové schéma výpočtu hmotnostních průtoků m_E a m_S.

Pro výpočet hmotnostního průtoku jsem použil model (viz. kapitola 2.1) z diplomové práce Mishukov, Artem "Mass Rate of Flow Virtual Sensor Inside the Air-Conditioning "[8]. Podle blokového schématu (obr. 14) jsem naprogramoval VS v kitu.

Naměřené teploty jsou v rozlišení na několik desetinných míst, proto je třeba použít datový typ *float*, případně *double*, aby mohly být hodnoty vypočítány co nejpřesněji a ne pouze s využitím celých čísel. Vybraný procesorový kit ale nemá vlastní FPU (Floating point unit), avšak *float* je podporován pomocí ABI (Application binary interface –), tedy softwarově. Výpočty pomocí ABI mohou být náročnější pro procesor, také mohou trvat déle, ale pro tento projekt stačí 2 výpočty hmotnostního průtoku (ze 4 teplot 2 průtoky – přiváděný a odváděný vzduch) za několik sekund.

3.6 Snímání napěťové úrovně ventilátorů

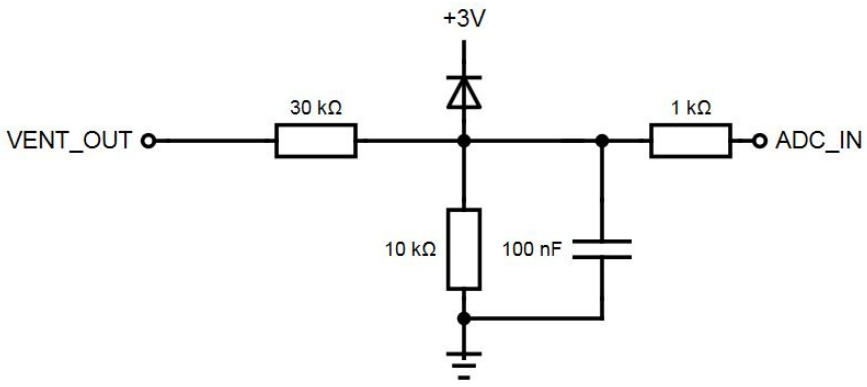
Hmotnostní průtok uvnitř výměníku je ovládán dvojicí ventilátorů v 6 napěťových úrovních. V důsledku působení prachu a dalších vlivů se ale teplota uvnitř výměníku zvýší a je nutné zvýšit také otáčky ventilátorů, které za ideálních podmínek odpovídají většímu hmotnostnímu průtoku, než v znečištěném prostředí. Proto můžeme díky odhadu hmotnostního průtoku detekovat, zda je jednotka zanesená a má tedy i vyšší spotřebu. Je tedy důležité sledovat napěťové úrovně, zda reálný průtok odpovídá nastavení. Pokud by se předpokládaný průtok (z napěťových úrovní resp. otáček je očekáván určitý průtok – tab. 3, 4) lišil od odhadovaného, který je měřen pomocí VS o více jak 10 %, tak lze říct, že jednotka je znečištěná a neefektivní.

otáčky [rpm]	průtok $[m^3 \cdot h^{-1}]$	napětí [V]
0	0	0
287,281436	698,753424	4,547246172
404,010238	1017,407848	6,394892875
483,035799	1242,228309	7,645752258
582,469891	1426,558961	9,219648924
631,770142	1624,530409	10

Tab. 3 Převod napěťových úrovní resp. otáček na hmotnostní průtok pro Ventilátor 1 (obr. 3) Použita naměřená data z experimentální jednotky.

otáčky [rpm]	průtok $[m^3 \cdot h^{-1}]$	napětí [V]
0	0	0
323,431058	827,807717	3,661464568
460,022815	1222,389282	5,207778276
611,103223	1482,898473	6,918113592
749,056042	1679,366879	8,479835469
883,337943	1738,034717	10

Tab. 4 Převod napěťových úrovní resp. otáček na hmotnostní průtok pro Ventilátor 2 (obr. 3). Použita naměřená data z experimentální jednotky.



Obr. 15 Zapojení napěťového děliče na ADC pin kitu.

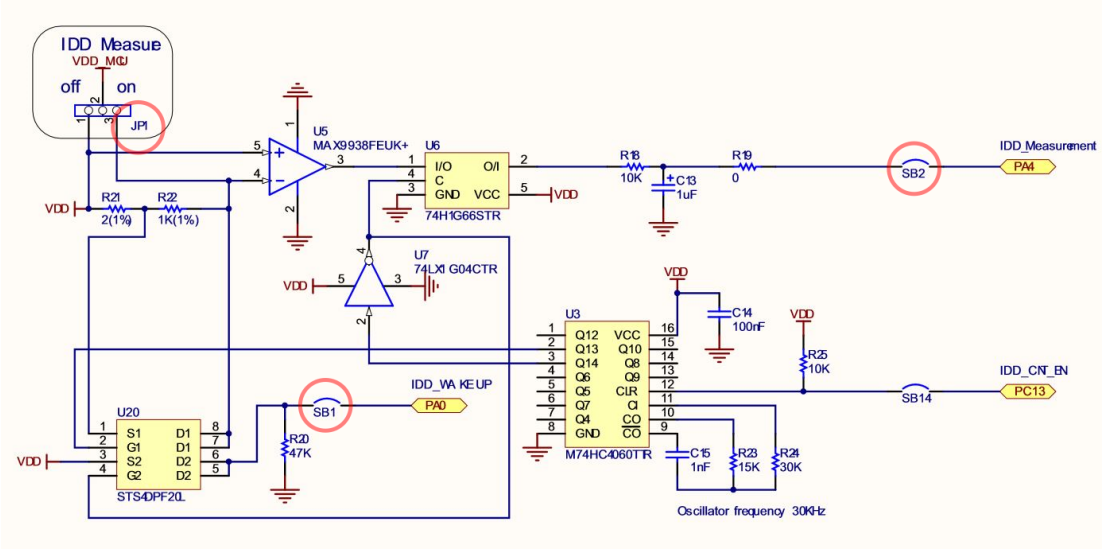
K měření napěťových úrovní mohu použít AD převodník přes napěťový dělič. Vzhledem k tomu, že ventilátory mají jen 6 nastavení, tak jejich měření není limitováno přesností ADC, naopak stačí 8bit rozlišení, tedy 256 diskrétních hodnot. Také není třeba

žádné rychlé odměřování, stačí mít nastaven nejdelší čas odměru, aby hodnota byla stabilní.

Napěťový dělič je nutné použít, protože AD převodník zvládne rozmezí 0–3V. Pro jistotu jsem použil dělič 3:1, tedy měřené napětí dělím 4. Případné přepětí může dosáhnout až 12V. Protože se může stát, že někdo špatně zapojí kabely, nebo se jen dotkne kontaktu (statická elektřina), použiji pro dělič odpory v 30 a 10kΩ. Také přímo k pinu kitu dám ochranný odpor kolem 1kΩ – viz obr. 15.

Po zapojení tohoto děliče jsem narazil na drobný problém. Tento kit má vestavěné měření proudu, které se v demo programu zobrazuje na displej. A pokud je zapojený displej, který je použit pro zobrazení všech hodnot, tak jsou volné jen právě 2 kanály ADC, jenže jeden z nich je napojený právě na tento měřič proudu, ale pro ventilátory jsou třeba oba.

Dle schématu se může zdát, že pokud se odstraní tzv. jumper propojka (JP1 – viz. obr. 16), tak to kanál neovlivní. Jenže tato propojka pouze zapíná obvod pro měření proudu. Problém se totiž nachází u spojek na piny procesoru (SB1 a SB2), kde se napájí část obvodu pro měření proudu (měřené napětí na ADC napájí část proudového měřiče) a vzniká tak úbytek měřeného napětí. Při použití děliče popsaného výše, byl úbytek napětí téměř o 0.8V. Bylo tedy nutné odstranit zapájené propojky (na obr. – SB1 a SB2 16) vedoucí jak na kanál, tak (pro jistotu) i na uživatelské tlačítko, které aplikace také využívá. Tím byly piny zcela odpojeny od obvodu pro měření proudu.



Obr. 16 Schéma měřiče proudu v kitu [4]. Pro použití kanálu 5 ADC bylo třeba odpájet propojky SB1, SB2 (označeno červeně).

3.7 Rozhraní USB

Měření hmotnostního průtoku je důležité především u systémů, kde není žádná senzorová síť. Proto by bylo dobré mít také možnost naměřená data ukládat a případně dále využít k vyhodnocení znečištění vzduchotechnické jednotky. K tomu může posloužit sériová komunikace přes rozhraní RS232. Toto rozhraní však není dnes tak běžné, jako standard USB, který je prakticky v každém počítači a není tak třeba použít žádnou redukci.

3.7.1 Virtual COM port

Vybraný kit je schopen komunikovat pomocí USB standardu tzv. Virtual COM portem. To je programový interface, který se v počítači chová, jako kdyby USB byl port pro sériovou komunikaci. Také se kit ve správci zařízení pod Windows jako COM port objeví.

Počítač vidí kit jako USB CDC (Communication device class – třída pro komunikační zařízení, nejsou zde žádné velké přenosy dat, jen probíhá výměna informací na nižších rychlostech – není využita plná kapacita USB).

3.7.2 PC aplikace pro komunikaci na USB

Abych si zopakoval programování v Javě, rozhodl jsem se v tomto jazyce naprogramovat PC aplikaci, která bude přijatá data z kitu zobrazovat a ukládat.

Pro aplikaci byly použity knihovny RxTx [9], které podporují sériovou komunikaci v Javě. Aplikace vytvoří vlákno pro příjem a zápis na zvolený sériový port (viz. sekce na webu RxTx [9] – Using RXTX -> Code examples -> Two way communcation with the serial port).

Také jsem vytvořil jednoduché grafické rozhraní, které usnadňuje práci s programem – stačí zadat COM port, potvrdit a aplikace každou přijatou zprávu z kitu okamžitě vypíše do okna. Veškerá vypisovaná data jsou také ukládána do *output.txt*

3.8 Ovládání a chod programu

3.8.1 Program na kitu

Po zapojení napájení provede program inicializace, vyhledá si senzory – počítá přímo se čtyřmi senzory. V případě, že je zapojeno méně senzorů, ohlásí na displeji chybu. Pokud je kit připojen k počítači, tak se provede také inicializace USB protokolu, aby bylo možné odesílat data také do PC.

Aplikace standardně běží bez zásahu uživatele, kdy každých 15 sekund odměří teploty ze senzorů (indikace pomocí modré LED), provede výpočet a získá také hodnoty z AD převodníku. Všechna data uloží do RAM a postupně zobrazuje na displej – při každém odměru se změní typ dat na zobrazovači. Také se rozsvítí zelená LED, pokud rozdíl mezi očekávaným průtokem skrze ventilátory a změřeným průtokem pomocí VS je větší jak 10% jako indikace poruchy.

Kontrola dat přímo na kitu se dá provádět pomocí uživatelského tlačítka, nebo externího tlačítka. Toto je také jediný možný zásah uživatele (kromě resetu). Tlačítko posunuje okamžitě zobrazované hodnoty, nemusí se tedy čekat na další odměr, aby se hodnota posunula.

3.8.2 PC aplikace

Po připojení USB kabelu je třeba zapnout PC aplikaci, aby mohla být všechna změřená data odesílána do PC, kde budou zobrazena v okně aplikace a také uložena to textového souboru. Při spuštění stačí zadat číslo COM portu, na kterém je kit připojen – viz. Správce zařízení ve Windows. Po potvrzení se již na obrazovce ukážou nová data po každém odměru. Výstup je také ukládán do složky s aplikací do textového souboru s názvem *output.txt*.

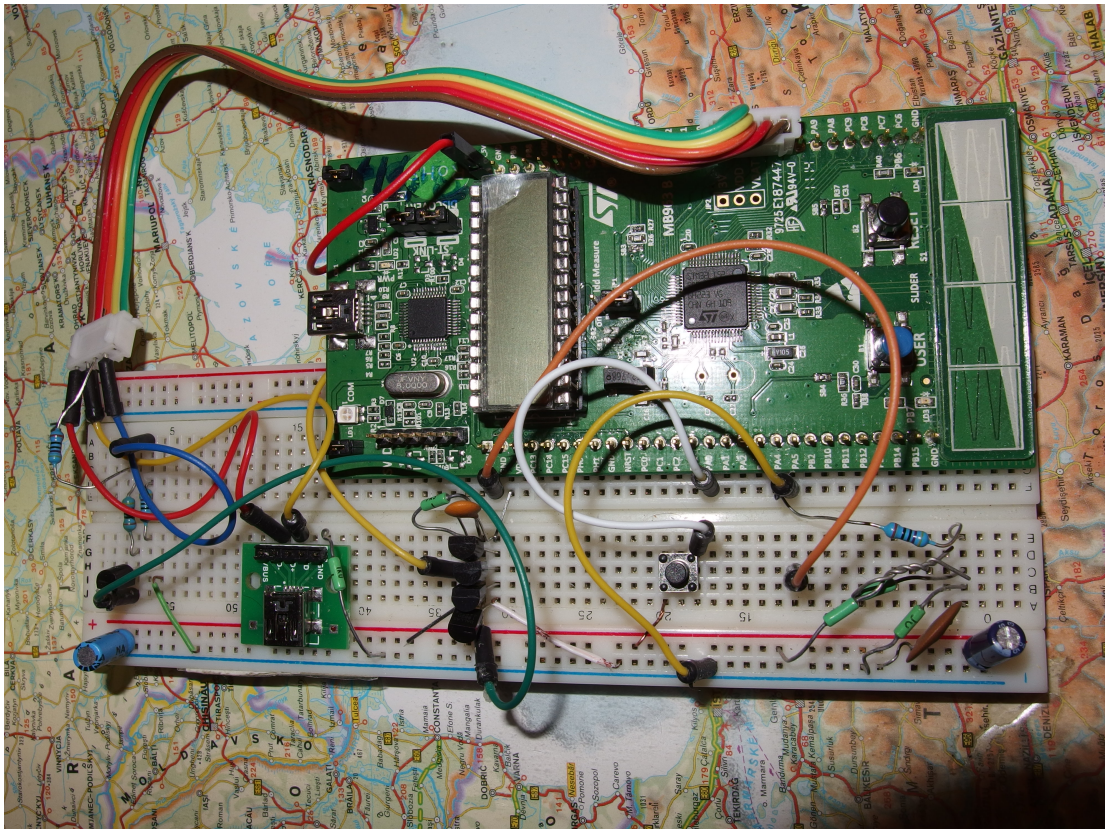
3.9 Realizace na pájivém poli

Aby bylo možné aplikaci snadno používat a vyvarovat se nechtěnému odpojení kabelů či přímému doteku s kitem (viz. zapojení na nepájivém poli, kde se každý kabel snadno odpojí – obr. 17), je třeba navrhnout zapojení na pájivé pole, které bude umístěno do plastové krabičky. Tím se také částečně zamezí znečištění prostředím a instalace je také snadnější.

Nejdříve je třeba vybrat krabičku takovou, aby i vnitřní rozměr odpovídal rozměrům kitu a krabička se tak nemusela upravovat už na začátku. Také aby kit "nelezl"z krabičky. Aby se vešla případná připojení na periferie, musí být krabička aspoň v jednom rozměru větší.

Vybral jsem plastovou krabičku, která je o 2 cm širší než kit, a vejdou se sem potom i periferie, které se budou připojovat pomocí šroubovacích svorkovnic, nebo v případě USB se použije konektor.

Pro USB použiji robustnější konektor (ten, který používají často tiskárny – USB typ B), který bude proti mikroUSB na kitu mechanicky odolnější, když se bude často připojovat a odpojovat.



Obr. 17 Realizace projektu na nepájivém poli.

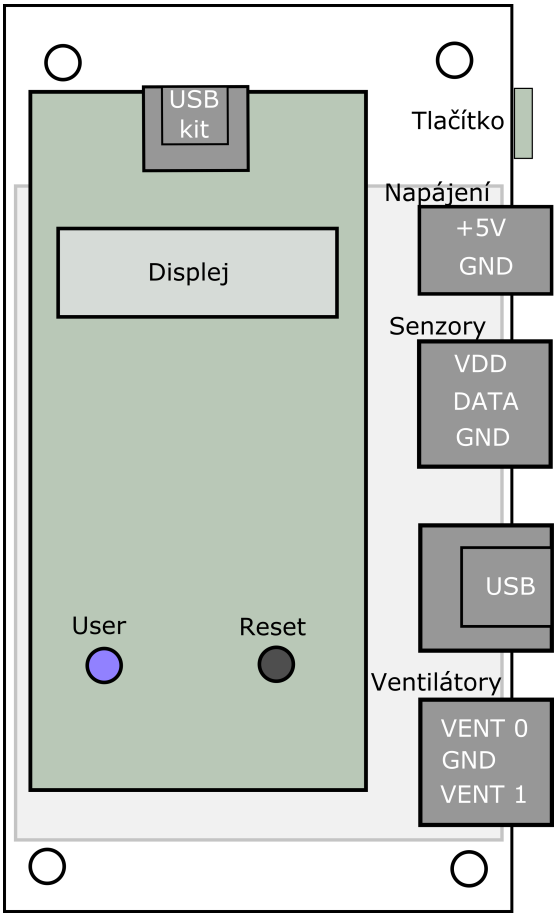
3.9.1 Mechanická konstrukce přístroje

Nejprve je třeba upravit krabičku tak, aby se do ní dal uchytit kit, který bude na pájivém poli. K tomu použiji distanční sloupky, aby pájivá deska nebyla přímo na plastu. Ty se uchytí jednoduše pomocí šroubků, takže stačí jen vyvrtat díry do krabičky a do pájivého pole na vybraná místa.

Zároveň je krabička relativně vysoká a je třeba, aby hrana displeje lícovala s krytem krabičky. Proto je nutné mít dostatečně vysoké distanční sloupky a vyříznout díru do krytu pro vestavěný displej kitu, aby bylo možné sledovat zobrazované údaje.

Tlačítka na kitu jsou přibližně ve stejné výšce, jako je displej. Proto je třeba vyvrtat díry i pro ně, aby je kryt nedržel stisknutá (například držené reset tlačítko by nedovolilo aplikaci vykonávat svou činnost). Zároveň je dobré mít možnost resetovat aplikaci, bez otevření krabičky, pokud dojde k nějaké chybě. Tlačítka budou také lícovat s krytem, aby nedošlo k náhodnému stisknutí resetu při manipulaci. Uživatelské tlačítko bude také zapuštěné, ale jeho funkci bude zastupovat paralelně přidané externí robustnější tlačítko na straně krabičky.

Mimo zobrazení pomocí LCD displeje používá aplikace také LED diody na kitu pro informaci o právě probíhajícím odměru, nebo rozdílu mezi průtoky vzduchu (kap. 3.6). Proto bude vyvrtán také otvor, aby bylo vidět na LED diody.



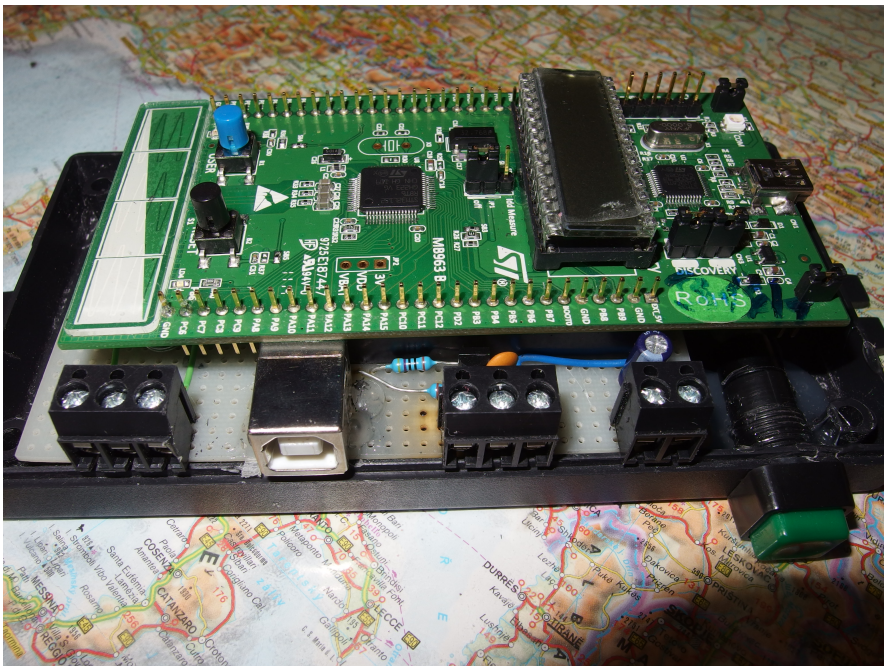
Obr. 18 Návrh rozložení svorkovnic, USB konektoru a tlačítka na pájivém poli.

Dále je třeba rozvrhnout zapojení na pájivém poli (obr. 18), které je nutné nejprve seříznout na velikost uvnitř krabičky a uchytit zmiňovanými distančními sloupky. Krabička byla zvolena o něco širší, a tak mohu všechny svorkovnice a konektory umístit na jednu stranu pájivého pole. Kit se do pole uchytí dutinkovými lištami, aby bylo možné ho v případě nutnosti snadno vyndat a nebyl pevně zapájen.

Na pájivé pole se tedy uchytí všechny svorkovnice, konektory a zapájí se také napěťový dělič před ADC (3.6), tranzistor a rezistory kolem datového pinu senzorů (3.4.1).

Protože svorkovnice musí přivádět kabely, stejně tak USB konektor, je nutné do boční strany krytu krabičky vyříznout patřičné otvory. Nakonec také otvor pro externí tlačítko, které bude také na této straně.

Původní napájecí zdroj a zároveň ladicí mikroUSB na kitu otvor mít nebude, protože není nutné k němu přistupovat tak často – pouze v případě nějaké závažné programové chyby, kdy by se musel program znovu nahrát. Napájení potom zajistí jedna ze svorkovnic a pro komunikaci je zajištěno USB.



Obr. 19 Finální realizace projektu na pájivém poli – víko krabičky odklopeno, možno zapojit napájení, senzory, ventilátory do připravených svorkovnic.

Obr. 20 Hotová krabička – vyříznutý otvor pro zobrazovací displej, tlačítka a LED. Externí zelené ovládací tlačítko.

4 Závěr

V této práci byl realizován virtuální senzor pro měření hmotnostního průtoku na vzduchotechnické jednotce umístěné na fakultě stavební ČVUT v Praze. Průtoky měřené VS jsou porovnávány s průtoky skrze ventilátory a v případě většího rozdílu (více jak 10 %) ohlásí chybu, takže je možné snadno poznat, zda jednotka pracuje efektivně či nikoliv.

VS je řízen mikrokontrolérem, který průtoky odhaduje (podle modelu) z teplot měřených na čtyřech místech uvnitř výměníku. Všechna data jsou zobrazována na displej přístroje, nebo mohou být také odesílána do PC přes USB rozhraní, kde mohou být dále snadno zpracovávána, neboť moje PC aplikace nejen zobrazuje, ale také ukládá data do textového souboru.

Tato realizace VS je především určena do jednotek, kde je jen minimální senzorová síť a usnadní tak kontrolu efektivního provozu vzduchotechnické jednotky.

Literatura

[1] Hanuš O. a Horyna V. "Experimentální vzduchotechnická jednotka pro vývoj a testování virtuálních senzorů a diagnostických metod". In: (2015).

[2] ST Microelectronics. *Datasheet*. 2013. URL: http://www.st.com/content/ccc/resource/technical/document/datasheet/66/71/4b/23/94/c3/42/c8/CD00277537.pdf/files/CD00277537.pdf/jcr:content/translations/en.CD00277537.pdf (cit. 04. 05. 2016).

[3] ST Microelectronics. *Reference manual*. 2013. URL: http://www.st.com/content/ccc/resource/technical/document/reference_manual/cc/f9/93/b2/f0/82/42/57/CD00240193.pdf/files/CD00240193.pdf/jcr:content/translations/en.CD00240193.pdf (cit. 04. 05. 2016).

[4] ST Microelectronics. *User manual – STM32L1 discovery kits*. 2013. URL: http://www2.st.com/content/ccc/resource/technical/document/user_manual/08/f8/63/f5/7b/3d/40/ff/DM00027954.pdf/files/DM00027954.pdf/jcr:content/translations/en.DM00027954.pdf (cit. 04. 05. 2016).

[5] Maxim Integrated. *Programmable Resolution 1-Wire Digital Thermometer*. 2015. URL: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf (cit. 04. 05. 2016).

[6] Majerle T. *OneWire library for STM32F4*. 2014. URL: http://stm32f4-discovery.com/2014/05/library-12-onewire-library-for-stm43f4xx/ (cit. 04. 05. 2016).

[7] Maxim Integrated. *Book of iButton Standards*. 2002. URL: http://pdfserv.maximintegrated.com/en/an/AN937.pdf (cit. 04. 05. 2016).

[8] Mishukov A. "Mass Rate of Flow Virtual Sensor Inside the Air-Conditioning Unit". Dipl. CTU FEL, 2015.

[9] RxTX. *RxTx*. URL: http://rxtx.qbang.org/wiki/index.php/Main_Page (cit. 04. 05. 2016).

Příloha A

Ukázky kódu

Ukázka A.1 Algoritmus vyhledání senzorů

```
uint8_t TM_OneWire_Search() {
uint8_t id_bit_number;
uint8_t last_zero, rom_byte_number, search_result;
uint8_t id_bit, cmp_id_bit;
uint8_t rom_byte_mask, search_direction;

/* Initialize for search */
id_bit_number = 1;
last_zero = 0;
rom_byte_number = 0;
rom_byte_mask = 1;
search_result = 0;

// if the last call was not the last one
if (!LastDeviceFlag) {
// 1-Wire reset
onewire_reset();
//if (onewire_reset()) {
/* Reset the search */
/*      LastDiscrepancy = 0;
LastDeviceFlag = 0;
LastFamilyDiscrepancy = 0;
return 0;
}*/

// issue the search command
onewire_write_byte(0xF0);

// loop to do the search
do {
// read a bit and its complement
id_bit = onewire_read_bit();
cmp_id_bit = onewire_read_bit();

// check for no devices on 1-wire
if ((id_bit == 1) && (cmp_id_bit == 1)) {
break;
} else {
// all devices coupled have 0 or 1
if (id_bit != cmp_id_bit) {
search_direction = id_bit;  // bit write value for search
} else {
// if this discrepancy if before the Last Discrepancy
// on a previous next then pick the same as last time
if (id_bit_number < LastDiscrepancy) {
```

```
search_direction = ((onewire_ROM_array[rom_arr_index-1][rom_byte_number]
                                        & rom_byte_mask) > 0);
} else {
// if equal to last pick 1, if not then pick 0
search_direction = (id_bit_number == LastDiscrepancy);
}

// if 0 was picked then record its position in LastZero
if (search_direction == 0) {
last_zero = id_bit_number;

// check for Last discrepancy in family
if (last_zero < 9) {
LastFamilyDiscrepancy = last_zero;
}
}
}

// set or clear the bit in the ROM byte rom_byte_number
// with mask rom_byte_mask
if (search_direction == 1) {
onewire_ROM_array[rom_arr_index][rom_byte_number] |= rom_byte_mask;
} else {
onewire_ROM_array[rom_arr_index][rom_byte_number] &= ~rom_byte_mask;
}

// serial number search direction write bit
onewire_write_bit(search_direction);

// increment the byte counter id_bit_number
// and shift the mask rom_byte_mask
id_bit_number++;
rom_byte_mask <<= 1;

// if the mask is 0 then go to new SerialNum
// byte rom_byte_number and reset mask
if (rom_byte_mask == 0) {
//docrc8(ROM_NO[rom_byte_number]);  // accumulate the CRC
rom_byte_number++;
rom_byte_mask = 1;
}
}
} while (rom_byte_number < 8);
 // loop until through all ROM bytes 0-7

// if the search was successful then
if (!(id_bit_number < 65)) {
// search successful so set LastDiscrepancy,LastDeviceFlag,search_result
LastDiscrepancy = last_zero;

// check for last device
if (LastDiscrepancy == 0) {
LastDeviceFlag = 1;
}

search_result = 1;
}
}

return search_result;
```

Ukázka A.2 Výpočet hmotnostního průtoku

```
void calculate_flow(float t2,float t3,float t8,float t9){
//t2,3,8,9 vstupni teplotni senzory
//std, mean a p[n] jsou dany z modelu
float ths, the, thes, ms, me;
if((t8-t2)==0||(t8-t9)==0) {
        me=-1;ms=-1;
        }
else{
        ths=(t3-t2)/(t8-t2);
        the=(t8-t9)/(t8-t2);
        thes=(t3-t2)/(t8-t9);


        if(ths>=the){
                ms=(ths-mean)/std;
                ms=p[0]*powf(ms,9)+p[1]*powf(ms,8)+p[2]*powf(ms,7)
                        +p[3]*powf(ms,6)+p[4]*powf(ms,5)+p[5]*powf(ms,4)
                        +p[6]*powf(ms,3)+p[7]*powf(ms,2)+p[8]*ms+p[9];
                me=ms*thes;
        }
        else{
                if(thes==0){
                        me=-1;
                        ms=-1;
                }
                else{
                me=(the-mean)/std;
                me=p[0]*powf(me,9)+p[1]*powf(me,8)+p[2]*powf(me,7)
                        +p[3]*powf(me,6)+p[4]*powf(me,5)+p[5]*powf(me,4)
                        +p[6]*powf(me,3)+p[7]*powf(me,2)+p[8]*me+p[9];
                ms=me/thes;
                }
        }
}
//pokud jsou hodnoty nerealne - male rozdily teplot
// kde model nefunguje, tak -1
if(me<-10000 || ms>10000 || ms<-10000 || me>10000) {
        me=-1;
        ms=-1;
}
data[6]=me;
data[7]=ms;
}
```